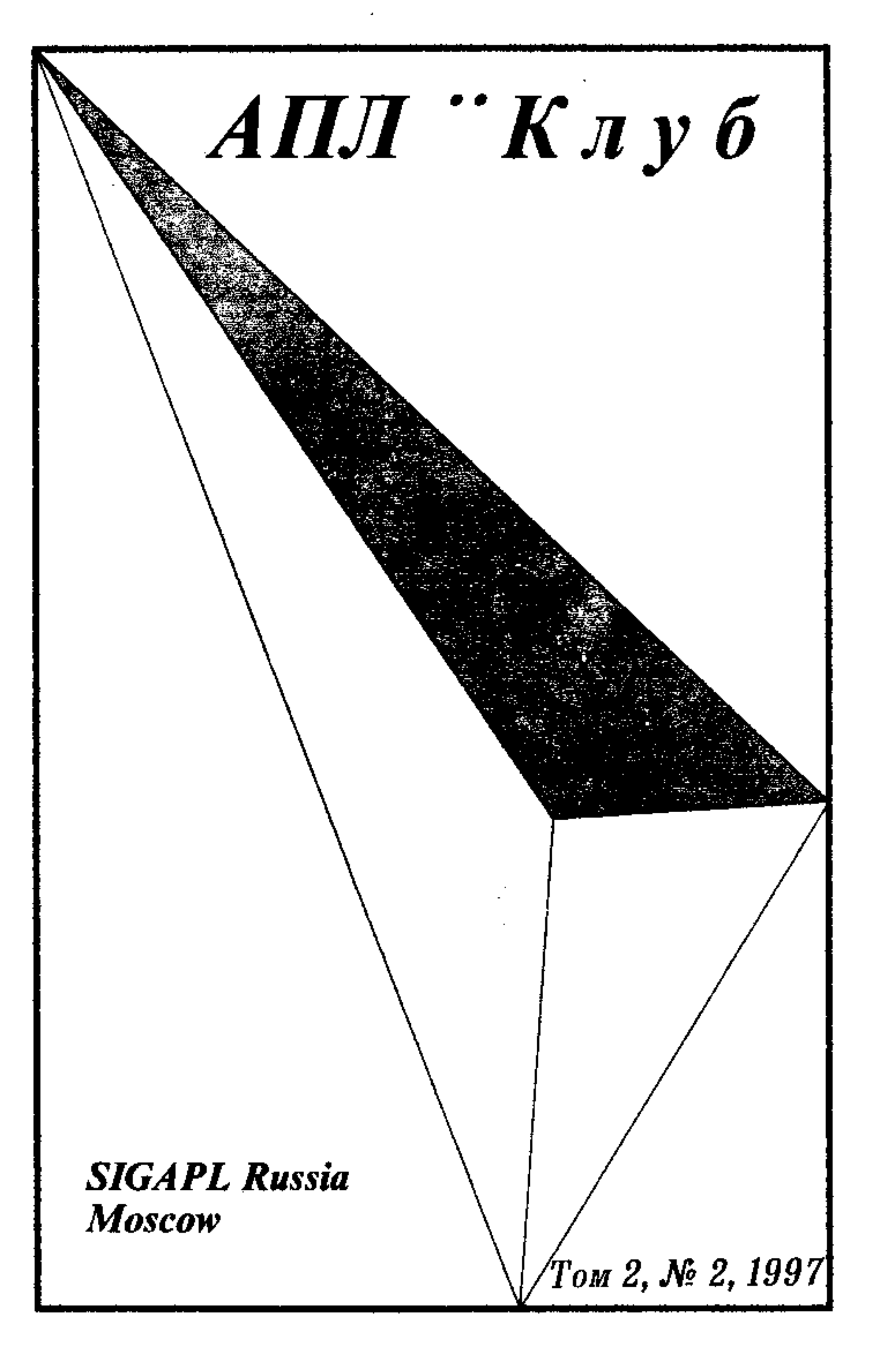

АПЛ ¨ Клуб
SIGAPL Russia
Moscow
Том 2, № 2, 1997

СОДЕРЖАНИЕ

Учредитель журнала "АПЛ "Клуб"-
Российская Ассоциация пользователей АПЛ (РосАПЛ)
Председатель РосАПЛ А.Ю. Бузин
Члены правления РосАПЛ:
А.И. Кононов, Б.А. Макеев, И.С. Нафтулин

Адрес Правления: *127434 Москва, Дмитровское шоссе 2, ЦНИИАИ, а/я 971 (для Макеева Б.)*
E-mail: *makeev@atom.ai.x-atom.net или buzin@ext1.ccas.ru*
Тел./Факс: *(095) 210-7783; тел. (095)313-4931*

CONTENTS

"АПЛ"Клуб" is the journal of
Russian association of APL users (RusAPL)
Chairman : Andrei Buzin
Members of the Board:
Alexei Kononov, Boris Makeev, Igor Naftulin

Mail address: 127434 Moscow,Dmitrovskoe sh., 2, CNIIAI, box 971
(for Makeev B.A.)
E-mail: *makeev@atom.ai.x-atom.net or buzin@ext1.ccas.ru*
Phone/Fax: *(095) 210-7783; Phone (095)313-4931*

Информация от Правления РосАПЛ

О конференции в Торонто 18-20 августа 1997

В начале 1997 года на сервере торонтского отделения SIGAPL наконец появилась информация о том, что традиционная международная АПЛ-конференция состоится и в 1997 году. Судя по этой информации, Торонтский SIGAPL решил взять на себя все хлопоты по организации этой конференции. По-видимому, у обычных организаторов - SIGAPL of ACM - возникли какие-то трудности.

Итак, объявлено, что конференция АПЛ97 состоится в Канаде, штат Онтарио, город Торонто, Ryerson Polytechnic University, Rogers Communications Centre 18-20 августа 1997 года. Приглашаются все, кто интересуется АПЛ, J и другими языками, ориентированными на обработку массивов.

Для участия конференции вы должны связаться по e-mail с организаторами конференции (см. ниже). В принципе, надо бы было еще выслать оргвзнос в размере $120, но, по-видимому, российские участники могут заплатить оргвзнос на месте (поскольку в России не принимают денежные переводы на почтовый адрес за границей).

Более полную информацию можно получить по адресу http://www.sigapl.mtnlake.com/sigapl/welcome.html в Internet или в Правлении РосАПЛ. По Internet можно даже получить стандартный буклет, посвященный этой конференции, предварительно "скачав" специальную программу Envoy Viewer.

Организаторы конференции особо отмечают дешевизну предстоящей конференции. Оргвзнос составляет всего $120 (намного меньше, чем на предыдущих конференциях). Цены в гостиницах - около $45 (надо полагать, что при желании можно найти что-нибудь подешевле).

К настоящему времени уже обнародована предварительная программа конференции:

Докладчик	Тема
Dyadic Systems	OLE и АПЛ
Dyadic Systems	Динамические функции в АПЛ
Dyadic Systems	АПЛ-вычисления в режиме "клиент/сервер"
Dyadic Systems	Программирование TCP/IP соединений
Insight Systems	ODBC и АПЛ
Eric Iverson - ISI / John Baker	Вычислительный сервер J/OLE
Timo Laurmaa	TCP/IP, HTML и АПЛ
Cliff Reiter - Lafayette College	J-искусство: Хаос и симметрия
Chris Burke - Strand	Графика и J
Chris Burke - Strand	Функциональная размерность в J для АПЛ-истов
Eric Iverson - ISI	Java и J
Mike Jenkins - NIAL	Курс Nial
Mike Jenkins - NIAL	Использование Nial для Web-приложений
Mike Jenkins - NIAL	Встроенные Nial-приложения
Chris Lee - SoftMed	Классы, определяемые пользователем в APL/Windows-программировании
Chris Lee - SoftMed	Объектно-ориентированные методы в APL/Windows-программировании
Eric Lescasse - Uniware	Тренажер APL+Win, объектно-ориентрованное программирование, MIDI, OCX и т.д.
Richard Levine	Введение в J для АПЛ-истов
Richard Levine	Один АПЛ-инструментарий
Ed Shaw - APL Group	Электронная коммерция, электронный обмен данными и Internet

B.Amos, G.Disney, D.Sorrey - Reuters	Обмен данными между Java-апплетами и юридическими АПЛ-приложениями
Andrei Buzin - RusAPL	Рекуррентный оператор в АПЛ
Murray Eisenberg - U of Massachusetts	J против Mathematica
John Heinmiller - Chalke / Eric Baelen - APL2000	PTS2000: История успехов АПЛ
John Heinmiller - Chalke	Составление финансовых отчетов с помощью АПЛ и PTS
Gary Mooney - Actel, et al.	Для чего хороши АПЛ и J?
Dennis Paproski - Reuters	APL IDE: Windows-интерфейс для АПЛ-приложений на мейнфреймах
Keith Smillie - U of Alberta	Компьютерное конструирование строения волн
Soliton Assoc.	АПЛ-система распознавания кодов
Soliton Assoc.	TimeSquare

На конференции будут проходить презентации фирм APL2000, Dyadic System Ltd., Soliton Associates Ltd.

Организаторы конференции предлагают следующие адреса для контактов:
Richard Procter, chair [rjp@interlog.com]
Eric Granz, finances [egranz@widow.aracnet.net]
Charles Chandler, program [chan@hookup.net]
Daniel Baronet, program [danb@vir.com]
Gaetan Godin, publications [gaetgodi@godin.on.ca]
Randy MacDonald, publications [randy@godin.on.ca]
Larry Moore, venue [lbm@soliton.com]
Tim Bishop, actuarial [tim_bishop@watsonwyatt.com]
Richard Levine, software [rlevine@aracnet.net]

Сравнительные характеристики современных версий АПЛ

Семинар 13 апреля 1997, Москва

Семинар состоялся в Вычислительном центре РАН. В семинаре приняли участие: Бузин А.Ю., Зуева А.В., Кононов А.И., Мешков Д.А., Нафтулин И.С., Соколов В.В., Ткачев А.Е.

Участники семинара поделились своими впечатлениями от работы с двумя разными версиями АПЛ - от Dyalog и от APL2000. Также были обсуждены вопросы, связанные с получением информации об АПЛ через Интернет.

Общее заключение от обмена мнениями по поводу двух версий АПЛ следующее: фирма APL2000 уделяет больше внимания вопросам программистского интерфейса, а Dyalog - развитию языка как такового. В DyalogАПЛ присутствуют такие языковые и системные элементы, которых нет в APLIII. Это в первую очередь относится к возможностям иерархической организации рабочих областей, оператора композиции и функционального присваивания.

Введение в функциональное программирование на J (часть 1).

(**Howard A. Peelle. *Introduction to Functional Programming in J (Part I)*.** Vector, Vol.13 No.1, pp. 20-29)

перевод Нафтулина И. С.

Это первая из трех предлагаемых Вашему вниманию публикаций. В первой части представлены основные принципы J в рамках стратегии функционального программирования, с примерами направленными на разработку программ, генерирующие простые числа. Во второй части (будет опубликована в следующем выпуске) будет продемонстрировано как непосредственно разрабатывать программы в J-стиле, а также будет предложена разработка программы вычисления простых чисел на более высоком уровне. В третьей части (в последующих выпусках) будут представлены другие варианты алгоритмов для итераций, рекурсий и операций с массивами, и в заключиние будут представленны мета-стратегии для разработки пакета программ и их документирования.

Мыслите в терминах массивов.

Прежде чем что-либо програмировать, надо понять, как будут формализованы данные для решения конкретной задачи. Этому этапу следует присвоить номер 0, чтобы показать, что он всегда присутствует именно перед началом реального кодирования. Нулевой этап включает обдумывание вопроса какие структуры наилучшим образом подходят для обработки рассматриваемых данных и формирования результата. В J, структуры данных это- массивы, т.е. отдельные величины (скаляры), списки (вектора) таблицы (матрицы), массивы трехмерные и более высокой размерности. Мыслить в терминах массивов- значит планировать обработку данных как единых массивов (где это возможно). В этой статье мы будем использовать в основном списки. Список чисел изображается с помощью пробела между ними. Например:

```
2 3 5 7 11                  NB. Список целых чисел.
```

Другие виды чисел (в этой статье в них нет нужды)представляются следующим образом: отрицательные числа представляются с помощью подчеркивания:_3. Действительные числа используют десятичную точку и лидирующий нуль, если необходимо, например: 0,5. Очень маленькие и очень большие числа представляются в научной нотации, например: 1e6 для одного миллиона. Для комплексных чисел используется инженерная нотация, например: 3j4 для 3 плюс 4 умноженное на квадратный корень из -1.

Заметим, что комментарии (им предшествует NB.) могут быть использованы для объяснений. Символьный список представляется с помощью одиночных кавычек. Например:

```
'primes'                    NB. Список символов.
```

Таблица изображается в виде прямоугольника из строк и столбцов. Например:

```
 2  3  4  5                 NB. Числовая таблица.
 6  7  8  9
10 11 12 13
```

(Способы создания таблиц будут описаны в частях II и III). Заметим, что в J нет необходимости декларировать и указывать тип данных. Данные различных типов могут быть размещены вместе в прямоугольной рамке, изображающей массив. Например:

primes	2 3 5 7 11

NB. Список двух прямоугольных элементов.
NB.Каждый из них содержит другие элементы.

Это позволяет представлять непрямоугольные массивы (см.[1]).

§1. Выбор примитивных функций

Первый этап программирования в J - это выбор примитивных встроенных функций необходимых для рассматриваемой задачи. J содержат свыше сотни примитивных математических функций. Сюда входят: арифметические функции: сложение, вычитание, умножение, деление, возведение в степень; функции сравнения; равно,неравно, меньше чем, больше чем и т.д.; логические функции и, или, не и т.п.; стандартные функции такие, как квадратный корень, случайные числа, целая часть; весьма полезные функции: возведение в квадрат, приращение, деление попалам и фикториал; такие сложные функции, как обращение матрицы и матричное деление для линейной алгебры и статистического анализа; функции перестановки и сортировки; тригонометрические и круговые функции; и множество других функций для обработки массивов, форматированного вывода, комплексных вычислений и т.п.

В этой статье приведены около дюжины примитивных функций языка J. (см. [1] для углубленного ознакомления). Заметим, что символы могут быть использованы сами по себе или в сочетании с точкой (.) или двоеточием (:) для обозначения различных функций: * (умножение), или *. (логическое и), или *: (возведение в квадрат).

В J функции называются "глаголами", и они используют "имена существительные" (данные) или "местоимения" (переменные) в качестве аргументов. Можно изучать поведение любого глагола, вводя выражения интерактивно, при этом отступ для ввода выдерживается автоматически и компьютор печатает результат на следующей строке. Например, рассмотрим глагол "остаток" (:), который использует существительные слева и справа:

```
   3:9              NB. 3 "остаток" 9
0                   NB. результат 0
   3:10             NB.  3 "остаток" 10
1                   NB. результат 1
   3:11
2
   3:12
0
```

Эти выражения можно было записать с использованием списка:

```
   3:9 10 11 12          NB. 3  "остаток " для списка
0 1 2 0                  NB. результат - список
```

Более длинный список показан ниже:

```
   3:0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 0 1 2 0 1 2 0 1 2 0
```

Примитивный глагол j. (называемый "целые") весьма удобен:

```
   j.13              NB. неотрицательные целые числа до 13
0 1 2 3 4 5 6 7 8 9 10 11 12
```

Эти глаголы можно использовать вместе:

```
   3:j.13                NB. остаток от целых чисел от 0 до 12.
0 1 2 0 1 2 0 1 2 0 1 2 0
```

Заметим, что здесь : является "идиадическим" глаголом (бинарная функция), который, подобно сложению (+), использует два входа (аргумента) (здесь скаляр слева, а список справа); а "целые" (j.) являются "монадическим" глаголом (унарная функция)

который использует один вход (аргумент) справа - подобно -4, или $\sqrt{4}$, или f(x) в общепринятых обозначениях. Все глаголы в J являются или монадическими, или диодическими.

Часто один и тот же символ используется, как в монодическом, так и диодическом варианте. Например: 7 - -4, т.е. 7 минус отрицание 4.

Далее заметим, что самый правый глагол выполняется первым, чтобы обеспечить вход для следующего глагола. Например, для вышеупомянутого примера, "отрицание" раньше "минуса" 7 - (-4), или (j.) раньше (:) в 3 : (j.13).

Фактически в J действует следующее общее правило: *каждый глагол использует результат выражения, записанного справа.* Это устраняет необходимость введения иерархии глаголов такого, как ("экспонента приоритетнее умножения и деления, которые в свою очередь приоритетнее сложения и вычитания"), но при этом возникают возражения (см. [2], где обсуждается реакция учителей математики на J нотацию). Тем не менее выход можно найти, используя круглые скобки, чтобы указать обычный порядок операций, как показано в этой статье.

§ 2. Выбор примитивных операторов

Кроме примитивных функций (глаголов), в J применяются операторы, которые модифицируют глаголы. С помощью операторов можно эффективно создавать новые глаголы, которые дают возможность программировать задачи более непосредственно. Например, "вставить" (Insert) (/) есть оператор "наречие", который вставляет глагол между последовательными элементами массива. Этот оператор использует глагол в качестве левого входа. Например:

```
      + / 2  3  5  7  11     NB. ("Сложение - Вставить")"plys- Insert"   список
28                           NB. сумма
```

Здесь глагол + является входом для /.

Два символа + / вместе создают новый глагол (называемый "суммой"), который воздействует на создание числа 2+3+5+7+11 и даст в результате 28. Программисты могут сравнить, как это делается в других языках программирования. Заметим, что J позволяет прямо использовать оператор вместо итераций с явным контролем за структурой. (См. часть 3 с дальнейшими обсуждениями).

Другие диадические глаголы также можно использовать вместе с "Insert". Таким образом программист может создавать новые глаголы. Например:

```
      */ 2  3  5  7  11
2310                                 NB. Произведение.
```

В действительности это просто повторение * столько раз сколько необходимо, чтобы вставить * между всеми элементами входа и затем вычислить 2*3*5*7*11 как 2*(3*(5*(7*11))).

Другие операторы будут обсуждаться по мере необходимости далее (полный список операторов привееден в [1]).

§ 3. Используйте параллельные вычисления

Естественная стратегия программирования на J подразумевает "параллельные вычисления", при которых глагол автоматически воздействует на каждый элемент массива. Обратимся к примеру глагола "остаток" (Remainder) (:) из § 1:

3 : 9 10 11 12 ` эквивалентен (3 : 9), (3 : 10), (3 : 11), (3 : 12), где скаляр слева взаимодействует с каждым числом справа. В общем случае, глагол одновременно, (параллельно) применяется к парам соответствующих элементов из левого и правого входов. Например, когда оба входа имеют одинаковое число элементов:

```
     1 2 3 4 : 9 10 11 12        NB. (1 :9). (2 :10). (3 : 11). (4 : 12)
0 0 2 0                          NB. Параллельное вычисление остатков
```

Это относится так же и к другим примитивным (встроенным) глаголам, как например:

```
     1 2 3 4 + 9 10 11 12
10 12 14 16                      NB. Параллельные суммы
     1 2 3 4 # 9 10 11 12        NB. (1 # 9), (2 # 10), (3 # 11), (4 # 12)
9 10 10 11 11 11 12 12 12 12     NB. Параллельное копирование
```

Здесь глагол # (Copy - копирование) создает копии каждого элемента. Этот глагол особенно полезен для выборки элементов:

```
     0 1 1 0 1 0 1 0 1 # 1 2 3 4 5 6 7
2 3 5 7
```

§ 4. Обозначайте вспомогательные функции и переменные

Программисты часто вводят вспомогательные функции (программы) и переменные для решения своих программистских задач. J использует =. для локального и =: глобального присвоения значений переменным (называемым "местоимениями"). Например:

```
p =: j. 12                 NB. p -целые числа 0 1 2 3 ... 11
```

Чтобы вывести на экран содержимое объявленой переменной, достаточно напечатать ее имя:

```
     p                     NB. что находится в p?
0 1 2 3 4 5 6 7 8 9 10 11
```

Объявленая переменная, конечно, может быть изменена, например с использованием простейшего глагола "increment" (шаг увеличения) > : , чтобы добавить 1 к каждому элементу списка:

```
     p=: >: p              NB. p является функцией "inkrement"
     p                     NB. что содержится в p ?
1 2 3 4 5 6 ... 12           NB. новое значение
```

Если запросить переменную, которой не было присвоено никакого значения, то это приведет к сообщению об ошибке:

```
     P                     NB. что содержится в P ?
value error                NB. в P не имеется никаких значений
```

Имеется более дюжины всевозможных сообщений об ошибках, включая "domain error" (ошибка области определения), "syntak error" (синтаксическая ошибка) и т. д.

J сообщает об ошибка, когда он не может интерпретировать выражение, но позволяет Вам ввести немедленно другое выражение.

Новые функции ("pro-verbs" - составные глаголы) в J определяются очень просто с помощью =. или =: , что означает "есть". Например, Вам необходимо использовать имя вместо символа для какого-либо глагола:

```
Remainder =. :             NB. Remainder есть :
```

Теперь это имя можно применить вместо символа : , например:

```
     3 Remainder 12        NB. тоже самое, что и 3 : 12
0
```

Это имя можно использовать, как функцию в любом выражении :

```
     (3 Remainder 12 ) = 0     NB. (3 : 12) равно 0 ?
1                              NB. Да
```

Заметим, что результат функции "Equal" -равно (=) равен или 1 (значение "истина") или 0 ("ложь").

Другие функции отношений, такие как "меньше чем" (<) или "больше чем" (>), так же возвращает булевый результат. (См. [1]).

Некоторым программистам может понадобиться использовать имя "mod" (сокращение по модулю) с переставленными входами. (Например, 12 по модулю 3). Это можно сделать следующим образом:

Mod = . : ~ NB. Mod есть Remainder - Commute

Символ ~ это наречие (называемое Passive или Commute), которое меняет местами входы глагола. Например:

12 mod 3 NB. 12 : ~ 3 (тоже самое, что 3 : 12)

0

Рассмотрим глагол "копировать" (#) из предыдущего параграфа, который использует 0 или 1 слева для выбора элементов из массива. Можно использовать # ~, чтобы выбрать элементы когда нули и единицы расположены справа. Удобно приписать этому глаголу имя:

If = . # ~ NB. If есть Copy-Commute (копировать-заменить)

Теперь можно использовать это имя как вспомогательную функцию:

1 2 3 4 5 6 7 If 0 1 1 0 1 0 1 NB. Выбрать числа, для которых If 1

2 3 5 7

§ 5. Составляйте (определяйте) собственные функции

J обеспечивает возможность создания традиционных многострочных программ с переменными и управляющими конструкциями (см. часть 3). Здесь мы создадим короткие (однострочные) программы без управляющих конструкций с помощью композиции функций. Это является ядром стратегии программирования в J. Более того, J позволяет определять программы ("pro-verb") без явных ссылок на входы или переменные. Это иногда называют "чистое функциональное программирование".

J предлагает несколько типов функциональных композиций. Начнем с оператора (@), называемого Atop (наверху), присоединяющего один глагол к другому - подобно математической функции. Например, создадим простой вспомогательный глагол для генерации положительных целых чисел:

Pos =. > : @ j. NB. Pos есть Inkrement Atop Integers
NB. (Дополнительные пробелы добавлены для лучшей читаемости)

"Atop" (@), есть оператор конъюнкции (conjunction), который требует два глагольных входа. Он использует глагол слева (здесь >:), для непосредственного воздействия на результат работы глагола справа (здесь j.), т. е. Increment после Integers. Теперь выполним новый глагол с одним входом:

Pos 12 NB. Положительные целые числа от 1 до 12
1 2 3 4 5 6 7 8 9 10 11 12

Другая удобная форма составления функций называется в J "вилка" (fork) используется, когда три глагола записываются ввиде отдельной последовательности.

Глагол в середине использует результат глаголов находящихся слева и справа. Например, (> : * < :) есть вилка, которая выполняет умножение в зависимости от результата Increment и Decrement. (< : является примитивным глаголом Decrement,

который вычитает единицу из его входа). Придадим этой функции ветвления следущее имя:

F = . > : * < :   NB. F суть Increment Times Decrement

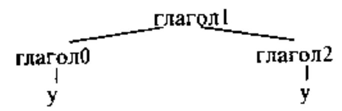
Испытаем с одним входом:

F = 4   NB. (> : 4) * (< : 4) или 5 * 3
15

В общем случае для одного входа y, ветвление

(глагол0 глагол1 глагол2) y

эквивалентно (глагол0 y) глагол1 (глагол2 y), это можно показать на диаграмме:



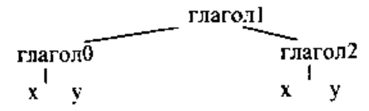
Ветвление можно использовать с двумя входами. Например:

F = . + * -   NB. F является Plus Tinus Minus
7 F 4   NB. (7 + 4) * (7 - 4)

В общем случае для двух входов x и y ветвление

x (глагол0 глагол1 глагол2) y

эквивалентно (x глагол0 y) глагол1 (x глагол2 y). Диаграмма:



Теперь рассмотрим вилку (: = 0 :), которое отличает глагол "равно" (Equal) с результатами "остаток" и "ноль". (: является глаголом - константа результат которого есть ноль- 0). Присвоим этому ветвлению имя:

Divides = . : = 0 :   NB. Остаток равно нуль.

Эта программа использует два входа. Например:

3 Divides 12   NB. (3 : 12) = (3 0: 12) или (3 : 12) = 0
1   NB. Истина (0 = 0)

Она работает так же со списочным входом:

(pos 12) Divides 12   NB. (1 2 3 ... 12 : 12) = 0
1 1 1 1 0 1 0 0 0 0 0 1   NB. Указывает на делимость элементов.

Давайте определим другую программу, использующую вилку внутри вилки:
Divisible = . Pos Divides]   NB. Divisible есть Pos Divides Input

Заметим, что] является примитивным глаголом, который возврашает свой собственный вход. Эта программа использует один вход:

Divisible 12   NB. (Pos 12) Divides 12
1 1 1 1 0 1 0 0 0 0 0 1

Глаголы - вилки могут объединяться в более сложные последовательности. Нижеприведенная программа использует другую "вилку":
Divisors = . Pos If Divisible   NB. делитель

Это эквивалентно следующему определению, в котором самая правая последовательность из трех глаголов объединяется с первым глаголом:

```
Divisors = . Pos If Pos Divides ]        NB. Divisors есть Pos If
                                          NB. (Pos Divides Input)
```

(См. часть 2, где объясняется как отразить на экране структуру такой программы.) Примеры использования:

```
    Divisors 12              NB. (Pos 12) If ((Pos 12) Divides 12)
1 2 3 4 5 6 12               NB. Делители для 12
    Divisors 13
1 13                         NB. Делители для 13
```

Эта программа является новым глаголом, который может быть объединен с другими глаголами. Например, примитивный глагол # Tally (число элементов) (# используется с одним входом) подсчитывает число элементов:

```
    # Divisors 12        NB. Tally Divisors of 12 (число делителей для 12)
6
    # Divisors 13        NB. Число делителей для 13
2
```

Теперь сравним результаты с числом 2:

```
    ( # Divisors 12) = 2     NB. (Tally Divisors 12) Equal 2 ?
0                            NB. Нет
    ( # Divisors 13) = 2     NB. Равно ли (Tally Divisors 13) 2 ?
1
```

Давайте объединим глаголы и создадим программу, обнаруживающую прошлое число (которое имеет точно два делителя):

```
Prime = . # @ Divisors = 2 :      NB. Простое число есть
                                  NB. (Tally Atop Divisors) Equal Two
                                  NB. Где глагол 2: возращает 2
```

Примеры использования:

```
    Prime 12        NB. Является ли 12 простым числом ?
0                   NB. Нет
    Prime 13        NB. Является ли 13 простым числом ?
1                   NB. Да
```

Теперь мы готовы расширить программу для генерации списка простых чисел. Это будет сделано в части 2, наряду с изучением следующих этапов программирования: тестированием, отладкой, моделированием, упрощением, оптимизацией и обобщением.

Литература.

[1] Iverson, K. E. *Introduction and Dictionry*, Iverson Software Inc., Toronto 1995

[2] Peele, H. A. "Teaching a Computer for Secondery Mathematics"

APL и J (2)- индексирование

(Chris Burke. "APL and J (2) -Indexing". VECTOR Vol. 13 No.2 pp 137- 143.
Перевод - И.Нафтулин, А.Бузин)

Серия статей "APL и J" является попыткой объяснить образ мышления, специфичный для программирования на J путем сопоставления APL и J с привлечением примеров. Термин APL будет использоваться в обобщенном смысле - используемые АПЛ-конструкции имеются в большинстве коммерческих версий APL.

В этой статье мы рассмотрим индексирование и индексированное присвоение в J.

Обозначения

Мы используем буквы V, M и A для векторов, матриц и массивов любой размерности соответственно. За этими буквами могут стоять числа, указывающие размеры массива. Например, V - любой вектор, V5 - вектор длиной 5, M45 - матрица размером 4 на 5, A245 - трехмерный массив размера 2×4×5.

Там где требуются конкретные значения, V5 будет определено как ⍳5, M45 как 4 5⍴⍳20. например:

```
      M45
 0  1  2  3  4
 5  6  7  8  9
10 11 12 13 14
15 16 17 18 19
```

Мы будем использовать терминологию APL, за исключением тех случаев, где без терминологии J нельзя обойтись. Так как J использует индексацию начиная с 0, будем предполагать, что ⎕io←0.

Индексирование

В APL для индексирования используются индексные выражения, заключенные в скобки. Такие выражения не поддерживаются в J; вместо этого в J используют функцию { (from). Сравним следующие выражения, для выборки второй строки из матрицы:

```
M[2;]          APL
2{M             J
```

Чтобы понять это нововведение в J, заметим, что индексные скобки в APL не согласуются с общепринятым АПЛ-синтаксисом. Действительно, индексирование - это двуместная функция, использующая в качестве аргументов данные и индексные указатели, однако оно не имеет обычного для двухместных функций синтаксиса - один аргумент справа, другой - слева.

Более того, вы не можете использовать индексирование в качестве обычной функции[1]. Например, если N является вложенным массивом, каждый элемент которого есть матрица, Вы не можете использовать индексные скобки, чтобы выбрать строку номер 2 из каждого элемента:

```
      N¨[2;]
SINTAX ERROR
```

[1] Именно поэтому в некоторые версии АПЛ введена функция индексирования, например, в DyalogAPL она обозначается как ⊃, а в APL2 - как ⌷. Надо заметить, что эта функция не обладает всеми возможностями, которое предоставляет обычное АПЛ-индексирование (*Здесь и далее - примечания переводчиков*)

Если Вы хотите сделать это, Вы должны определить функцию для индексирования матрицы:

```
[0]  r←ndx rowndx mat
[1]  r←mat[ndx;]
```

Для выборки второй строки из матрицы M следует использовать выражение 2 rowndx M, а для выборки второй строки из всех элементов N - выражение 2 rowndx¨N.

Соответствующие выражения в J имеют вид:

```
2 { M
2 { each N
```

Как видите, { является просто функциональной формой индексных скобок. Поскольку { является обыкновенной функцией, ее можно использовать в функциональных выражениях, как показано выше.

Идея { проста, но не тождественна индексной функции ⌷ в APL2.

Заметим, что если мы пожелаем в APL выбрать второй слой из трехмерного массива, нам потребуется изменить функцию rowndx, но в J выражении все останется без изменений.

Приведем еще несколько примеров:

```
item2=. 2 & {              функция возвращающая второй элемент

item2  M45
10 11 12 13 14

item2  A324
16 17 18 19
20 21 22 23

mrow=. { & M45             функция возвращая строки M45

mrow 3 2
15 16 17 18 19
10 11 12 13 14
```

Индексирование с левым аргументом

Левый аргумент может иметь различные формы:

1. Это могут быть одно или несколько целых чисел, которые выбирают соответствующие элементы из правого аргумента. Отрицательные числа выбирают от конца, например:

```
   2 _1  { M45             вторая и последняя строки M45
10 11 12 13 14
15 16 17 18 19
```

Часто бывает удобным явное указание размерности функции { , например:

```
   2 _1 {"1 M45            второй и последний столбцы M45
2   4
7   9
12 14
17 19
```

2. Левый аргумент может быть одним или большим количеством списков, заключенных в скобки, каждый элемент из которых есть целые числа, которые выбирают по соответствующим осям:

```
    M45[1;3]                    APL
8
    (< 1 3) { M45               J
8
```

Ниже приведенные выражения позволяют разделить индексы:

```
     M45[(1 3)(2 4)]            APL2 и Dyalog APL
8 14

    (< 1 3; 2 4) { M45               J
8 14
```

3. Списки в скобках из левого аргумента могут сами содержать списки, каждый элемент так же выбирает по соответствующим осям:

```
     M45[1;3 4]                 APL
8 9

( < 1 ; 3 4 ) { M45                  J
8 9
```

Заметим, что в J это последнее выражение позволяет разделить индексы, где каждый индекс может иметь различное число элементов.

Следующий пример имеет два индексных указателя:
первый выбирает строку 1, столбцы 3 и 4;
второй- выбирает строку 2, столбец 3. Эти результаты объединяются в матрицу, дополняя вторую строку нулем.

```
     (( 1 ; 3 4 ) ; 2 3 ) { M45
 8  9
13  0
```

Выбор всех элементов вдоль оси

В APL когда ось опущена, выбираются все элементы этой оси. Это не то же самое, что использование пустого индекса:

```
      M45[;2]    выбор всех элементов второго столбца
2 7 12 17

    M45 [ '' ; 2 ]    не выбрать ни одного элемента из второго столбца
```

В J пропуск оси обозначается с помощью пустого вектора в скобках:

```
     ( < ( <'' ) ; 2 ) { M45        выбор всех элементов второго столбца
2 7 12 17

     ( <'' ; 2 ) { M45  не выбрать ни одного элемента из второго столбца
```

Вместо пустого вектора удобно использовать "a:" (функция *ace*, порождающая вложенный пустой вектор):

```
   ( < a: ; 2 ) { M45          выбор всех элементов второго столбца
2  7  12  17
```

В общем случае, аргументы заключенные в скобках используются как индексы для выбора соответствующих элементов. Пустые скобки выбирают все. Ниже, выбираются первая и последняя строки из столбцов 2 и 3:

```
   ( < ( < 0 _1 ) ; 2 3 ) { M45
 7  8
12 13
```

Индексированное присваивание

Так же, как и для индексирования, синтаксис индексированного присваивания в АПЛ не соответствует стандартному синтаксису. Эта операция требует трех аргументов- индекса и двух массивов, один из которых обязательно должен быть представлен своим именем. Имя одного массива и индексы указываются слева от присваивания, другой массив указывается справа:

```
M[2;]←V                 заменить строку 2 из M на V
```

Индексированное присваивание не является функциональным выражением. Его явным результатом является присваиваемое значение (в данном примере V), а изменение M является побочным эффектом.

Напротив, в J индексированное присваивание является частью языка. Эта операция состоит из двух шагов:

- индексы задаются как аргументы наречия } (*amend*), которое является двуместной функцией, воздействующей на свои два аргумента в соответствии с индексами.

- затем эта функция применяется к двум аргументам- данным. Элементы левого аргумента заменяют элементы правого аргумента согласно индексам.

Например, чтобы заменить строку 2 из M на V:

```
V 2 }M
```

Порядок выполнения следующий:

V (2 }) M   2 }является функцией, которая связана с элементом 2.

Чтобы заменить строку 2 для каждого элемента вложенного списка матриц N :

```
(< V ) 2 } each N
```

Заметим, что индексированное присваивание в J не требует присваивания и обновления элементов, поэтому нет необходимости в именовании аргумента. Изменение M осуществляется следующим образом:

```
M=. V 2 } M
```

Аргументы для Amend

Индексные аргументы для } симметричны индексным аргументам для {. Например,

```
   2 { M45                  выбор строки 2
10 11 12 13 14

   ( 100* i. 5 ) 2 } M45    изменение строки 2

0   1   2   3   4
5   6   7   8   9
0 100 200 300 400
15 16  17  18  19

   ( < 1 ; 1 2 3 ) { M45    выбор строки 1 столбцов 1 2 3
6 7 8

   100 200 300 ( < 1 ; 1 2 3 ) } M45    заменить строку 1 столбцов 1 2 3
0   1   2   3   4
5 100 200 300   9
10 11  12  13  14
15 16  17  18  19
```

Селективное присваивание

Современный APL может заменить часть массива, выбранного с помощью выражения заданного слева от знака присваивания. Эта операция называется селективным присваиванием. Например, заменить все 2 в V на 20:

```
      V←1 2 2 3 1 2
      ((V=2)/V)←20
      V
1  20  20  3  1  20
```

Подобные средства в J обеспечиваются использованием функцией amend, которая позволяет создать требуемые индексы. Например:

```
      bx 1 0 1 1 0 1 1        утилита bx возвращает индексы единиц
                              булевого вектора
0 2 3 5 6

      f=. bx @ (2 : = ])      f возвращает индексы элементов правого
                              аргумента, равных 2.

      20 f } 1 2 2 3 1 2      использует f для замены 2 на 20
1 20 20 3 1 20
```

Выборка (Fetch)

Новой функцией в J3.02 является {:: *fetch*, которая в случае бинарного использования x `fetch` y выбирает из *y* подобласть указанную в *x*; при этом выбор на каждом уровне производится с помощью { .

Например:

```
[A=. 1 2 3 ; 4 5 ; i. 4 5
```

```
┌─────┬─────┬──────────────────┐
│1 2 3│4 5  │ 0  1  2  3  4    │
│     │     │ 5  6  7  8  9    │
│     │     │10 11 12 13 14    │
│     │     │15 16 17 18 19    │
└─────┴─────┴──────────────────┘
```

(2 ; _1 _1) fetch A   выбор из второго элемента элемента из последней строки и столбца.

19

(2 ; < 2 ; 2 3 4) fetch A   выбор из второго элемента элементов, имеющих индекс 2 ; 2 3 4

12 13 14

Такая выборка действует аналогично проникающей индексации в Dyalog APL, за исключением того, что в J на выбираемый элемент не наложено ограничение, чтобы он был скаляром:

```
      A[(2(2 2))(2(2 3))(2(2 4))]
12 13 14
```

Литература

Iverson, K. E. *J Introduction and Dictionary*, Iverson Software, 1996.

АПЛ-операторы и J

Norman Thompson (Vector v.13#1)
(перевод, предисловие и послесловие А.Бузина)

Предисловие от переводчика История АПЛ довольно драматична. Кеннет Айверсон, отец АПЛ отказался от старшего сына и полностью посвятил себя младшему - J'ю. Тем не менее, эти два языка до сих пор сосуществуют рядом, и нельзя сказать, что кто-то из них одержал окончательную победу. По-видимому, у J есть свои достоинства и недостатки по сравнению с АПЛ. Я не знаком с J, но первые впечатления, вынесенные мной из чтения статей в Vector, уже сформировались, и я хотел бы их высказать.*

Во-первых, в J перенесена из АПЛ идеология работы с массивами как с целостными объектами, а также идеология конструктивного (а не явного) описания массивов. Во-вторых, сохранено большое количество примитивных функций. Это - то, что я могу сказать о сходстве J и АПЛ.

Теперь о различиях. Бросающееся в глаза различие - отказ от нестандартного АПЛ-алфавита. Айверсон, по-видимому, учел многочисленные упреки по поводу АПЛ-алфавита. Однако, в целях сохранения лаконичности языка, автору J пришлось ввести двухсимвольные обозначения для примитивов. J не стал более читабельным чем АПЛ, а может быть, стал даже менее читабельным.

Насколько я понял, в J введено понятие "функциональной размерности", то есть самой функции приписана размерность, определяющая размерности возможных аргументов, направление действия функции и размерность результата. Кроме того, расширены соглашения о возможностях конкатенации массивов разных размерностей.

*Основное отличие заключается в том, что J сильно продвинулся в сторону развития примитивных операторов. Этот язык, в отличие от классических языков программирования, рассматривает в качестве объектов операций не только данные, но и функции, причем сами операции могут порождать как данные, так и функции. Результатом выражения в J может быть **имя функции**. Так например, в J введены примитивные операторы (т.н. "союзы"), которые могут порождать новую функцию, связывая имена двух других.*

*Эти усовершенствования сделали J еще более изящным языком программирования по сравнению с АПЛ. Возможность построения функций из функций в режиме командной строки, несомненно, впечатляет. Это приближает J к математическим обозначениям, позволяет говорить о том, что языки программирования могут описывать работу не только в дискретных числовых пространствах, но и в функциональных. Заметим, однако, что подобные усовершенствования являются скорее **лингвистическими**: они усовершенствуют саму запись, но не отражаются на эффективности вычислений. С другой стороны, конструкции языка J не являются ни привычными (для классического программиста), ни простыми в понимании (для меня).*

Мне представляется, что главная заслуга создателей АПЛ (а вслед за ним и J) заключается в том, что он позволил программистам отказаться от программистской рутины - описаний структур данных, циклов, форматирования ввода/вывода. К сожалению, при этом пришлось пожертвовать эффективностью вычислений. Можно сказать, что классические языки (C++, Паскаль и др.) пошли по пути повышения эффективности конечного машинного кода, упуская из виду вопрос об эффективности написания первичного программного кода. АПЛ и J пошли в противоположную сторону. Истина, как всегда, лежит посередине. Хотя для более основательных утверждений о перспективах развития АПЛ, J и их соотношении с классическими языками программирования требуется более серьезное знакомство с J, я думаю, что центральной задачей для APL/J направления является в настоящее время не совершенствование их языковых конструкций, а попытка создания квазикомпилятора (то есть компилятора избранных кусков АПЛ-кода) и совершенствование взаимодействия с другими системами. (Конец предисловия)

J заслуживает внимания АПЛ-программистов с точки зрения использования операторов. Разработчики J сделали использование операторов ключевым моментом языка программирования.

Грубо говоря, АПЛ-"функция" соответствует понятию "глагол" в J. Понятие оператора J разделяет на два: "наречие"(adverb) (унарный оператор) и "союз"(conjunction)(бинарный оператор). Союзы подразделяются на три типа - "существительное-существительное", "существительное-глагол" и "глагол-глагол". Таким образом, всего рассматривается 4 типа конструкций, соответствующих АПЛ-операторам. Понимание этого факта может существенно отразиться на стиле вашего АПЛ-программирования. Я буду для краткости использовать термин "операторное программирование"; я утверждаю, что стиль операторного программирования существенно увеличивает общность программного кода[2].

Перед тем, как отдельно рассмотреть все 4 типа операторов, поясню на примере, что я подразумеваю под "операторным стилем программирования". Рассмотрим стандартную для начинающего АПЛ-иста программу вычисления среднего:

```
[0]  Z←mean R
[1]  Z←(+/R)÷ρR

     mean 1 4 5 6
4
```

Используя операторный стиль и держа в уме способ "вилки" (fork) из J, перепрограммируем эту задачу, располагая ÷/ в качестве самой левой операции выражения, которое производит над R два преобразования - сначала ρ, а затем +/. Объединим эти две функции с помощью оператора AND, который применяет функции-операнды к аргументам-данным[3]:

```
[0]  Z←{L}(P AND Q)R          ⍝Оригинал подправлен
[1]  →(0=⎕NC 'L')/L1
[2]  →0 Z←(L P R)(L Q R)
[3]  L1:Z←(P R)(Q R)
```

Теперь запишем новый текст для функции, вычисляющей среднее:

```
[0]  Z←mean1 R
[1]  Z←÷/+/ANDρR

     mean1 1 4 5 6
 4
```

(Опытный читатель заметил, что перед результатом появился дополнительный пробел, который означает, что он имеет глубину 2; эта техническая деталь легко может быть устранена).

Известно, что среднее арифметическое не является единственным определением

---

[2] Да, судя по нашему опыту, это действительно так. То, что автор называет "операторным стилем" позволяет использовать имена функций для порождения новых имен функций без использования редактора функций. Классические языки программирования не позволяют этого делать. Результатом функции в классических языках программирования могут быть только данные, но не имена.

С одной стороны, такая конструкция очень красива и еще больше приближает АПЛ к языку математики. Например, приятно сознавать, что функция второй производной функции f в АПЛ записывается почти так же как в математике: f D D. С другой стороны подобный стиль наверняка добавляет дополнительные проблемы к вопросу компилируемости. Кроме того, к этому нетрадиционному стилю, в частности к приоритетам операций (функций и операторов) не так просто привыкнуть. *(Прим. переводчика)*

[3] Интересно отметить, что версия DyalogAPL предоставляет возможность сымитировать способ "вилки" с помощью оператора композиции. Например, вычисление среднего арифметического может быть записано следующим образом: `Mean← ÷/∘(⍎¨)∘('+/' 'ρ'∘(,¨))∘⊂∘⍕`. В общем случае, выражение (f2 X)F(f1 X) эквивалентно `F/∘(⍎¨)∘('f2' 'f1'∘(,¨))∘⊂∘⍕`. Этот факт говорит о том, что DyalogAPL движется в сторону J.

среднего: другими часто встречающимися определениями является среднее геометрическое и среднее гармоническое. Эти определения различаются тем, что среднее арифметическое вычисляется от данных, предварительно преобразованных с помощью разных функций. Преобразуем функцию вычисления среднего, превратив ее в оператор:

```
[0]  Z←(P MEAN1)R
[1]   Z←÷/+/ANDρP R

      +MEAN1 1 4 5 6
4
```

Гармоническое среднее вычисляется следующим образом:

```
      ÷÷MEAN1 1 4 5 6
2.4742
```

, а геометрическое среднее:

```
      *⍟MEAN1 1 4 5 6
3.3098
```

В последних двух выражениях с левого края стоит имя функции, обратной к следующей за ней. Мы могли бы переписать наш оператор вычисления среднего как бинарный:

```
[0] Z←(P MEAN Q)R
[1]  Z←P÷/+/ANDρQ R

      +MEAN+ 1 4 5 6
4
```

Аналогично вычисляются гармоническое и геометрическое средние:

```
÷MEAN÷ 1 4 5 6
*MEAN⍟ 1 4 5 6
```

Теперь запрограммируем функцию отклонения от среднего. Начнем с такой[4]:

```
[0] Z←adev R
[1]  Z←∊|R-+MEAN+R
```

Используя эту функцию совместно с функцией MEAN, можно получить абсолютное отклонение следующим образом:

```
      +MEAN adev 1 4 5 6
1.5
```

Теперь давайте подумаем: абсолютное отклонение является лишь одним из видов отклонений, другим видом является квадратичное отклонение, чье среднее является дисперсией. Итак, опять вернемся к оператору:

```
[0] Z←(P DEV)R
[1]  Z←∊P R-+MEAN+R

[0] Z←sq R
[1]  Z←R*2

      +MEAN |DEV 1 4 5 6
1.5
      +MEAN sq DEV 1 4 5 6          ⍝5
3.5
```

Функция sq сама по себе является частным случаем экспоненты, можно, например, определить оператор экспоненты:

```
[0] Z←(P EXP)R
[1]  Z←R*P

      +MEAN(2 EXP)DEV 1 4 5 6
3.5
```

---

[4] По-видимому, автор использовал не DyalogAPL (возможно, APL2). Судя по всему унарный примитив ∊ используется автором как "разрушить". В DyalogAPL он имеет такое значение только при ⎕ML=3. То же замечание относится и к тексту оператора DEV. Мы даем коды автора в оригинальном виде, но следует учесть, что, если вы используете другую версию АПЛ, то их надо немного изменить.

[5] В DyalogAPL можно было бы обойтись и без функции sq: `+MEAN (*∘2) DEV 1 4 5 6`

```
      +MEAN(3 EXP)DEV 1 4 5 6
¯4.5
```

Последнее выражение, как легко понять вычисляет момент третьего порядка.
Если выразить суть "операторного программирования" одной фразой, то можно сказать: "Зачем определять несколько функций, если можно сразу определить их семейство?".
В следующем разделе приведены примеры каждого из 4-х типов операторов, упомянутых в начале статьи.

1. Операторы, используемые как наречия

Простейшими примерами являются операторы редукции и внешнего произведения (но НЕ внутреннего, которое является оператором типа глагол-глагол). Примером оператора-наречия является также описанный выше оператор EXP. Вы можете подумать, что между выражениями 2 EXP 3 и 2⋆3 нет никакой разницы. Между тем, это совсем не так. Выражение (2⋆)3 является синтаксически неправильным, в то время как выражение (2 EXP)3 не только эквивалентно выражению 2 EXP 3, но и иллюстрирует механизм исполнения этого выражения, называемый в компьютерных науках "кэррингом" (currying) по имени американского логика H.B.Curry. Грубо говоря, этот механизм заключается в замене операции с двумя переменными операцией с одной переменой путем включения другой переменной в саму операцию[6]. Таким образом, 2 EXP является производной функцией, которая может быть применена[7], например, к вектору 1 2 3 для получения вектора 2 4 8.
Другим примером кэрринга является:

```
  [0] Z←(P LOG)R
  [1]  Z←P⍟R

      10 LOG 2
0.30103
```

Описанный выше оператор DEV также является примером оператора-наречия. Он модифицирует действие операнда-функции с помощью указания на то, что между ним и аргументом операции должна быть вставлена операция вычисления отклонений от среднего.

2. Операции типа "существительное-глагол"

Простым примером оператора этого типа[8] является RPT1, который производит обращение к функции заданное количество раз[9]:

```
  [0] Z←L(P RPT1)R;I
  [1]  I←0 ⋄ Z←L
  [2] L1:→(R≤I←I+1)/0
  [3]  Z←P Z
  [4]  →L1

      2(2 EXP)RPT1 5
65536
```

(пятый член последовательности 2,4,16,256...).

---

[6] В таком случае, типичным кэррингом являются вторая и третья форма оператора композиции в DyalogAPL.

[7] Главная прелесть, однако, не в том, что эту функцию можно применить к данным (к данным всегда можно применить и выражение "2⋆"), а в том, что эту функцию можно использовать как аргумент оператора (например, оператора дифференцирования).

[8] Не ясно, почему оператор RPT1 отнесен к типу "существительное-глагол". В соответствии с определением, данным в начале статьи, этот тип относится к союзам, которые являются <u>бинарными</u> операторами. Однако RPT1 - унарный оператор.

[9] В оригинале вместо строчек 3 и 4 используется строка →L1 Z←P Z, которая, по крайней мере в Dyalog 7.2 является синтаксически неверной. Можно предположить, что автор работает с АПЛ2 от IBM. Об этом же говорит и использование примитива ∊ в смысле "разрушить".

Вариацией на эту тему является оператор RPT2, который повторяет обращение к функции до тех пор, пока разница между двумя последующими результатами вычисления этой функции больше чем правый операнд Q:

```
[0]  Z←L(P RPT2 Q)R
[1]   Z←L P R
[2]  L1:→(Q>|Z-R)/0
[3]   →L1(Z←L P R←Z)
```

Например, известный алгоритм вычисления квадратного корня может быть записан с помошью функции

```
[0]  Z←L sqrtstep R
[1]   Z←0.5×R+L÷R
```

и оператора RPT2 следующим образом[10]:

```
      9 (sqrtstep RPT2 0.001) 1
3.000000001
```

Другим примером является оператор RPT из работы [1].

3.Операции типа "глагол-глагол"

Выше мы уже встречались с операторами из этого класса. Это были операторы AND и MEAN. Операторы этого типа позволяют образовать некоторую композицию из двух функций: детальное описание этой композиции и представляет собой тело оператора. Приведем еще пример - оператор аналогичный наречию языка J с тем же именем:

```
[0]  Z←(L)(P WITH Q)R
[1]   →(0=⎕NC'L')/L1
[2]   Z←(Q L)P(Q R)
[3]   →0
[4]  L1:Z←(P R)(Q R)
```

Приведем пример использования этого оператора (моделирование умножения с помощью сложения логарифмов):

```
      10⋆2+WITH(10 LOG)3
6
```

4.Операции типа " существительное -существительное"[11]

В качестве примеров можно привести несколько способов использования операторов. При использовании кэрринга эти способы позволяют увеличить количество аргументов функции до трех или четырех. Простым примером является формула сложных процентов $A(1+R)^N$, которая по существу включает три аргумента. Кэррингуя величину 1+R, можно использовать оператор:

```
[0]  Z←L(P CI)R
[1]   Z←L×P⋆R

      100 (1.1 CI) 2 3
121 133.1
```

---

[10] Пример приведен к правильному синтаксическому виду на Dyalog 7.2.

[11] Оба примера, приведенные в этом разделе, по-моему, неудачны. Они представляют не бинарные, а унарные операторы, а поэтому по приведенной классификации относятся к классу наречий! Пример с оператором CI вообще принципиально не отличается от примера с оператором LOG (см. выше). Иллюстрацией типа "существительное-существительное" может быть, например, оператор

```
[0]  r←A(L CI1 R)N
[1]   r←A×(L+R)⋆N
```

, обобщающий опрератор CI или опрератор "суммирования главного минора n×m" матрицы X:

```
[0]       r←(n SumMinor m)X
[1]   X←n m↑X
[2]   r←+/,X
```

Или, например, формула Герона для вычисления площади треугольника по трем сторонам может быть запрограммирована так[12]:

```
[0]  Z←L(P TRIAREA)R
[1]  Z←(×/(0.5×+/L,P,R)-0,L,P,R)*0.5

      3(4 TRIAREA)¨5 6     ⍝[13]
6 5.33
```

Является ли операторное программирование более совершенной формой программирования? Думаю, да. В первой части этой статьи мы разработали небольшую прикладную систему, включающую четыре оператора - AND, MEAN, DEV, EXP - и ни одной функции.

Использование операторного подхода подразумевает осмысление двух вопросов: а)что является сутью данной операции и б)насколько данную операцию можно обобщить. Например, содержание операторов, описанных выше, показывает, что:

суть MEAN это ÷ +/ и ⍴

суть DEV это EXP[14] и MEAN

суть EXP это *

и каждый из этих операторов является обобщением соответствующих функций.

Выше я попытался показать, что если программист склонен к обобщениям, то операторы возникают естественным образом. Использованные примеры были достаточно просты, чтобы их сложность не скрывала суть используемых аргументов. Читатель, интересующийся более сложными иллюстрациями, может обратиться к работе [1].

<u>Послесловие от переводчика</u> Использование АПЛ-операторов действительно предоставляет возможность программирования многих вычислительных задач в АПЛ-стиле: кратко и адаптировано к интерактивным экспериментам. В каком еще языке программирования <u>имя</u> второй производной функции f будет выглядеть как D D f, то есть почти также, как в математике? Именно операторы придают АПЛ такую лаконичность, хотя односимвольные имена многих примитивных функций также играют немаловажную роль. Использование определяемых операторов позволяет обобщать довольно широкие классы задач (хотя, возможно, многим программистам это и не понравится - все-таки, это черта математической, а не программистской культуры - см., например, правило 2.2 в книге А.Голуба "C&C++. Правила программирования") . Тем не менее, для вычислителя, программирующего разнообразные задачи и ценящего свое время, использование операторов может оказаться настоящей золотой жилой. Посмотрите, как с помощью одного оператора (рекуррентного оператора) можно облегчить себе жизнь при программировании огромного количества задач вычислительной математики (см. мою статью в предыдущем номере "АПЛ"Клуба")! Поэтому идея статьи Н.Томсона оказалась мне очень близкой.

Однако, хочу сделать одно критическое замечание по поводу второй части статьи, - той, где дается классификация операторов.

Операторы в АПЛ действительно можно подразделить на два непересекающихся класса - унарные и бинарные. Однако бинарные операторы нельзя классифицировать по трем подклассам, как в J (глагол-глагол, глагол-существительное и существительное-существительное). Это верно по крайней мере для DyalogAPL. Дело в том, что одно и то же имя оператора можно применять с операндами разного типа.

Действительно, по тексту оператора RPT2 (см. выше) ничего нельзя сказать о том, к какому подклассу он относится. Его можно применять, используя в качестве операндов

[12] Вот уж совсем неудачный пример: стороны треугольника в некотором роде "равноправны" (они одинаково влияют на площадь) и их естественно делать единственным аргументом функции!

[13] Думаю, что в тексте ошибка: там пропущен оператор ¨.

[14] В оригинале почему-то вместо EXP стоит *; наверное, опечатка.

как имена функций, так и имена данных. Н.Томсон относит этот оператор к типу "глагол-существительное " и приводит соответствующий пример. Но вот примеры применения этого же оператора с двумя "существительными":

```
      ⍳(2 RPT2 3)4
1 2 4
```

или с двумя "глаголами":

```
      9 (sqrtstep RPT2 F001) 1
3.00000001
```

, где

```
  [0]   r←F001
  [1]   r←.001
```

Утверждение о том, что бинарные операторы классифицируются по трем типам в DyalogAPL неверно даже для примитивных операторов. Действительно, оператор композиции может быть применен как в виде "глагол-глагол", так и в виде "глагол-существительное".

Следует заметить, что возможность использования данных в качестве операнда оператора добавляет гибкости интерпретатору, но порождает большие трудности при формальном описании языка. Практически эта возможность приводит к слиянию понятий оператора и функции. Действительно, становится неясным, является ли унарная ⍳ оператором или функцией, поскольку, примененная к данным (например к символьному скаляру '×') она порождает имя функции.

В классическом учебнике по АПЛ2 (Brown, Pakin, Polivka) авторы так и не смогли дать строгого определения оператора. На стр. 78 они пишут, что операторы применяются к функциям. Однако, на стр.208 подчеркивается, что символ "/" всегда (даже, когда слева стоит массив) обозначает оператор. При определении синтаксиса описания определяемого оператора (стр.89) на операнды оператора также не наложено никаких ограничений. Реализация АПЛ2 следует именно этому принципу: "1 0 1/" является в АПЛ2 правильным именем функции. Такой подход хорош, конечно, тем, что можно сказать: "символ "/" является именем оператора". Однако возникает вопрос, почему, например, символ "." является не только именем оператора? И что хорошего, если один и тот же символ обозначает совершенно разные операторы (редукции и репликации) в зависимости от контекста?

В документации к DyalogAPL (стр.21 Language Reference) дано более четкое определение оператора. DyalogAPL рассматривает символ "/" как имя бинарной функции (репликации) или унарного оператора (редукции) в зависимости от контекста. Хотя этот подход такой же запутанный, как и в АПЛ2 (лучше всего было бы определить другой символ для репликации; но такова уж традиция), он имеет то достоинство, что делает все примитивные операторы, за исключением оператора композиции (которого, кстати, нет в АПЛ2) операциями только над функциями[15]. Сам по себе оператор композиции, нарушающий[16] стройность концепции оператора, мог бы быть разделен на два: суперпозицию функций и кэрринг (см. статью), причем последний можно было бы для объявить особой операцией - ни функцией, ни оператором.

Как видите, концепция оператора в АПЛ (появившаяся вместе с АПЛ и являющаяся одним из краеугольных камней АПЛ) довольно сложная и по-разному трактуется разными производителями. Думаю, что это было одной из причин, побудивших Кеннета Айверсона скорректировать эту концепцию в J.

Библиография

[1] Technical Note on Confidence Limits, N.Thomson, The Random Vector, v.12, No.3,pp.98-106, April 1996

---

[15] В версии DyalogAPL 7.2 , к сожалению, будет вычисляться выражение 2((4).×)3, но это, видимо, мелкая ошибка разработчиков, которые забыли проверить тип операнда, если аргументы являются скалярами.

[16] Мы уже говорили о том, что ⍳ вообще нарушает все, что угодно.

А.Кононов

APL в Internet

Введение

«Дорвавшись» до Internet, автор, естественно, попытался найти в этой вселенской информационной свалке какую-нибудь информации об АПЛ. И мы не обманулись в своих надеждах. Но начнем по порядку ...

Поскольку тему Internet мы в нашем журнале еще не освещали, обзорно рассмотрим базовые моменты этой «напасти» и термины, которые сейчас вошли в оборот.

Опуская вопросы, «как подключиться к Internet» и «кто за это будет платить», остановимся на том, **как и чем найти** информацию об АПЛ в Internet, и рассмотрим основные **Internet-ресурсы**, связанные с АПЛ, т.е. места, где эта информация чаще всего аккумулируется. Читателю следует иметь ввиду, что наш опыт работы в Internet еще не столь велик, и сам читатель, пользуясь нашей «наводкой» и располагая временем, возможно, проникнет в более глубокие информационные пласты, чем это удалось нам.

Для начала скажем, что Internet включает в себя огромное множество разнотипных ЭВМ, связанных между собой информационными каналами разного типа, и хранящих на своих накопителях постоянно обновляющуюся информацию в различных форматах - данные для непосредственного восприятия (т.е. документы, изображения, звук, видео), компьютерные программы, и систематизированные сведения (базы данных), поэтому из Internet можно получить не только оперативную информацию о предстоящих **семинарах и конференциях** по АПЛ, но и переписать («скачать») **статьи** по АПЛ и даже бесплатные **интерпретаторы и рабочие области** с АПЛ-программами, готовыми к применению. (Не забывайте при этом, что вместе с программами в Ваш компьютер из сети могут попасть и вирусы!).

В мире существует множество компьютерных сетей с разными названиями и самого разного масштаба - от внутрифирменных сетей для решения задач производства и сбыта до глобальных коммерческих, почтовых и справочных сетей. Internet охватывает большинство из них, поэтому Internet можно рассматривать как сеть сетей. В настоящее время в Internet входят несколько глобальных информационных подсистем:

- информационная сеть WWW;
- служба архивов FTP;
- информационная система Gopher;
- система телеконференций Usenet;
- система баз данных WIAS;
- электронная почта E-mail;
- справочная подсистема X.500;
- адресные книги WHOIS.

Для взаимодействия всех этих разнотипных ресурсов естественным образом возникло организованное подмножество серверов Internet, которое в обиходе называют World Wide Web или **WWW** - «Всемирная Паутина». WWW сейчас обеспечивает интерактивный режим доступа к информации миллионам пользователей, электронную почту **E-mail** и перекачку файлов с **FTP**-серверов (компьютеров, хранящих файлы для копирования по запросу удаленного заказчика на его компьютер по протоколу File Transfer Protocol -FTP). Все это хозяйство принадлежит самым различным учреждениям и фирмам, часть из которых (Internet-провайдеры) зарабатывают себе на жизнь только

продажей услуг связи с Internet. Помимо Internet существуют и автономные host-компьютеры (серверы) или группы серверов, к которым можно подключиться по телефону и получить информацию или списать файл - это системы типа **BBS** (Bulletin Broad System - система текстового оповещения), например CompuServe и America Online. Многие из BBS уже фактически стали частью Internet, т.е. для доступа к ним можно зарегистрироваться у ближайшего провайдера Internet. В России услуги Internet сейчас начинают предоставлять как крупные сети, ориентированные ранее на пересылку электронной почты и файлов, так и новые. Вот некоторые из отечественных Internet-провайдеров: Релком, Демос, Совам Телепорт, Россия-Он-Лайн, РОСНЕТ, ГласНет, СИТЕК, Интернет/Россия, RUHELP/Radio-MGU.

Оперативная информация в Internet предоставляется сетью WWW в виде **Web-страниц**. Web-страница - это небольшой документ, иногда размером с экран дисплея, созданный по гипертекстовой технологии средствами языка HTML (HyperText Markup Language - язык гипертекстовой разметки) обычно с помощью специального редактора. Текст Web-страницы содержит выделенные шрифтом и цветом слова и фразы, являющиеся указателями на другие страницы. Самое замечательное, что эти страницы-ссылки могут находится как на этом же диске, так и на дисках других компьютеров, разбросанных по всему миру. Помимо текста, Web-страница может содержать мультимедийные элементы - статические (ссылки на GIF-файлы) и двигающееся изображения, а также иметь звуковое сопровождение - и все эти элементы также могут храниться в виде отдельных файлов в произвольных местах WWW.

Обычно основная задача Web-страницы - представить в Internet какую либо организацию или фирму. АПЛ"Клуб возможно в ближайшее время заведет свою страничку в Internet, например, с адресом http://www.aplclub.ccas.ru. Точный адрес Вам будет сообщен позднее, а на примере этого можно изучить структуру URL-адреса в Internet (Universal Resource Locator - универсальный указатель ресурса). Адрес читается задом наперед (как мы привыкли в АПЛ): ru - Россия, ccas - ВЦАН (Вычислительный Центр Академии Наук), aplclub - наша страничка, www - серверы группы WWW, http - указатель протокола обмена с сервером (http - ***h**yper**t**ext **t**ransfer **p**rotocol*). Формат адреса базируется на формате файлов в ОС UNIX, широко используемой на серверах в Internet. В структуре адреса «хвостик» определяет либо страну (ru или su - Россия (СНГ), ca - Канада, us -США, fr - Франция, uk - Великобритания) либо группу организаций (gov - правительственные, com - коммерческие, org - некоммерческие, edu - образовательные и исследовательские). Группа имен, разделенных точками, формируют имя домена - узла сети, представляющего сервер (host-компьютер), постоянно соединенный с Internet - и определяеют положение сервера в иерархической структуре сети.

Совокупность Web-страниц создает документ для интерактивной работы, при которой входящие в него элементы просматриваются не последовательно, а в порядке «хода мысли» читателя. С точки зрения системы, такой документ (если он находится на одном компьютере) представляется совокупностью вложенных каталогов и файлов. Например имя http://www.acm.org/catalog/sigs/sigapl.html представляет собой адрес странички SIGAPL, хранящейся в виде файла с именем sigapl.html вместе со страничками для других SIG-ов в каталоге sigs, являющегося подкаталогом каталога catalog в группе каталогов организации ACM (Association for Computing Machinery) на серверах WWW.

Помимо Web-страниц в Internet возможен доступ к специальному виду оперативного взаимодействия **Новости (News)** или **конференции** через сеть Usenet News, но при условии, что Ваш Internet-провайдер предусмотрел такую услугу для своих клиентов.

Коллекции файлов находятся на FTP-серверах, куда их могут помещать одни пользователи, а загружать (копировать на свои компьютеры) - другие, примерно так, как мы копируем с дискеты файл на винчестер Norton Commander-ом. Например файл с адресом ftp://archive.uwaterloo.ca/languages/apl/fonts/2741 содержит шрифт (фонт) для

АПЛ-символов из набора, совместимого со шрифтом с телетайпа IBM 2741, и этот шрифт можно скопировать с FTP-сервера университета Ватерлоо в Канаде.

Для облегчения удаленному пользователю навигации в коллекциях файлов длительного хранения (в архивах) созданы сотни индексных серверов, на которых штатом администраторов формируются индексы файлов с потенциальным спросом. Старейшей такой сетью является поисковая сеть **Gopher** («Суслик» или 'go fer' - рыщущий в поисках чего-либо). Серверы Gopher работают на ОС UNIX и пользователю предоставляется механизм поиска файлов на основе иерархической файловой структуры UNIX. Для обращения к серверам Gopher в адресе имеется специальное указание, например gopher://veronica.scs.unr.edu/11/veronica дает полный адрес каталога, через который можно осуществить поиск в некотором Gopher-архиве.

Если пользователю точно известно местонахождение файла, он может из Internet через механизм **telnet** непосредственно «залезть» в интересующий его компьютер, например для работы с базой данных, поддерживаемой только в данном месте (на данном site).

Наконец, для обслуживания электронной почты (**E-mail**) в Internet существуют почтовые серверы, а для поиска почтовых адресов пользователь Internet может воспользоваться базой данных адресов электронной почты (mailDB).

Инструмены Internet

Если для работы с файловой системой MS-DOS наиболее популярной программой является Norton Commander, то на рынке средств для работы в Internet сейчас лидируют два **браузера** (программы просмотра Web-страниц) - Microsoft Internet Explorer версии 3 (уже появилась 4) и Netscape Navigator версии 3. Первому из них активно покровительствует сама Microsoft, включая его в штатный состав ОС Windows NT 4.0 и в расширение Windows-95 Plus. Фактически сейчас каждая из этих программ стремится объединить или автоматизировать доступ к нескольким инструментам Internet, существовующих в многочисленных реализациях самостоятельно (например программа Eudora для работы с E-mail, Archie и WS-FTP для работы с FTP, News Express для NEWS, WSGopher для Gopher).

Работа с браузером, например с MIE (Microsoft Internet Explorer), как правило труда не представляет - сиди и листай Web-страницы. Основная трудность - как быстро выйти на нужную информацию вследствие колоссальных масштабов информационных ресурсов. Здесь на помощь приходят **поисковые системы в Internet** (**SE** - search engines - машины поиска) и автоматизированные **каталоги** (**DIR**- directories), базирующиеся на мощных суперЭВМ или группах производительных серверов.

Допустим, нам нужно найти информацию о языке АПЛ в Internet.

Предварительно можно сузить район поиска, отнеся искомую информацию к какому-то классу (наука, компьютеры, языки программирования) или задав ее вид (Web-страницы, архивные файлы, новости). В зависимости от подхода к поиску, можно воспользоваться либо поиском **по тематике** через DIR или **по контексту** документов через SE.

Пусть мы решили выйти на АПЛ по тематике. Остается обратиться к серверу, работающему в режиме DIR. В Internet имеется несколько довольно популярных DIR-систем, но наиболее известными являются YAHOO (http://www.yahoo.com) и INFOSEEK (http://www.infoseek.com). Они совмещают в себе свойства DIR и SE системы.

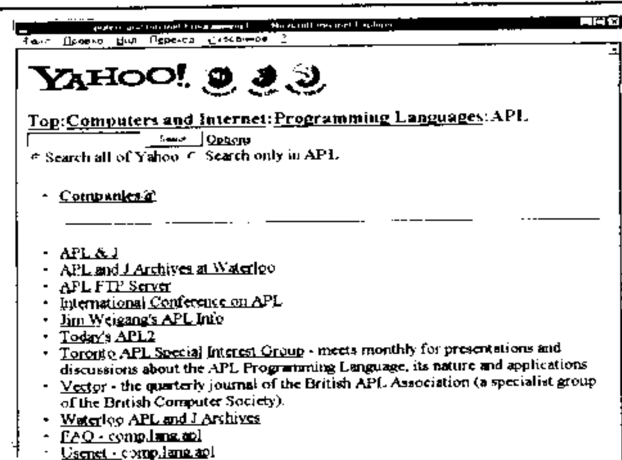


Рис.1

Выбрав любую из этих систем, например YAHOO, мы увидим первую страничку, содержащую перечень основных разделов - Искусство, Бизнес, Наука ... , среди которых находим подходящий по теме: Компьютеры и Интернет (здесь - перевод; фактически львиная доля информации в Internet дается по-английски, поэтому мы так и продолжим). Нажав на эту строчку попадаем на страничку с описанием разделов, связанных с компьютерной тематикой и находим Programming Languages. «Проваливаясь» в этот подраздел получаем список языков программирования и нажимаем на APL. В итоге мы проделали следующий путь в YAHOO: Computers and Internet: Programming Languages: APL и получаем страничку, показанную на Рис.1.

Далее мы выходим на более конкретную информацию, например запрашиваем перечень всех компаний, имеющих отношение к языку АПЛ, нажав на **Companies@**.

Здесь нам YAHOO дает указатели на Web-страницы двух «китов» АПЛ - американскую фирму APL2000 (продолжатель линии APL*PLUS Manugistics и STSC) и английскую Dyadic (линия DyalogAPL). «Наезжая» мышью на их имена мы внизу экрана увидим соответствующие WWW-адреса: http://www.apl2000.com и http://www.dyadic.com . Запомнив, их можно в дальнейшем использовать для прямого выхода на соответствующий сервер.

Теперь можно продолжить нашу экскурсию и войти в эти странички. Заметим, что сильное впечатление оставляет оформление Web-узла фирмы Dyalic. Вам будет предложено «скачать» специальный АПЛ-фонт и настроить Windows для непосредственного чтения фрагментов АПЛ-кода. Странички содержат множество иллюстраций, например объясняющих работу с таблицами (grids) (Рис.2).

Поиск в Internet по контексту, выполняется с помощью какой-либо из SE, позволяет получить подборку информационных ресурсов разного типа по заданному вами запросу. При этом, чем точнее сформулирован запрос, тем меньше вы получите «пустой породы».

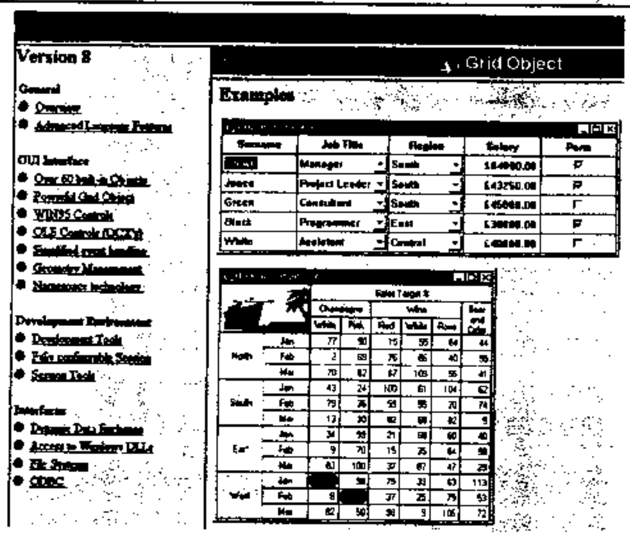
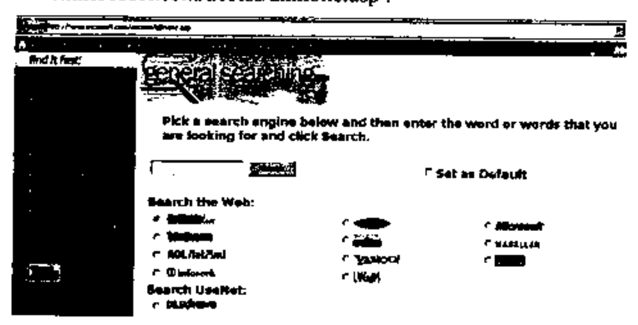


Рис.2

Наиболее известными машинами поиска (средствами которых нам удалось найти информацию об АПЛ) являются системы AltaVista (http://www.altavista.digital.com), HotBot (http://www.hotbot.com), Excite (http://www.excite.com).

Список популярных DIR и SE можно найти, нажав в MIE на кнопку «Поиск» или по адресу http://home.microsoft.com/access/allinone.asp :



Вот, например, как выглядит (Рис.3) запрос к AltaVista выдать все что есть по «APL Language» (здесь вы видите и фрагменты рекламной информации). Здесь же установлен режим поиска по Web-страницам и вывод в стандартном виде.

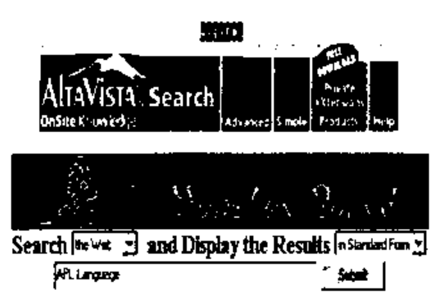


Рис.3

В ответ система нашла более 40000 документов, выдав на страничке заголовки первой десятки из них с соответствующими адресами. Первая же ссылка в этом списке вывела нас на страничку (Рис.4), близкую по содержанию Рис.1 («так все запутано в этой паутине!») и имеющую адрес

http://www.dn.net/yippee/comp_lan_apl.html.

Продолжая навигацию вручную и выбрав первый пункт, мы попадаем в Web-страницу SIGAPL (Рис.5).

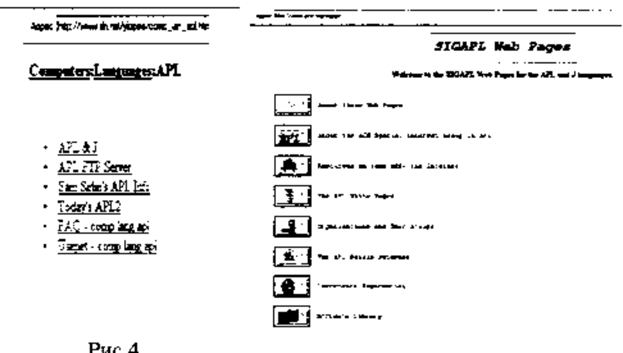


Рис.4

Рис.5

Третья позиция в этой странице открывает нам систематизированный перечень основных ресурсов внутри Internet и за его пределами, имеющих отношение к языку АПЛ. Запомним полный адрес этого списка:

http://www.acm.org/sigapl/Resources/HomePage.html

Отсюда мы узнаем следующие полезные адреса:

- Наиболее частые вопросы об АПЛ (FAQ) - http://grover.jpl.nasa.gov/~sam/pub/apl.faq
- Сервер SIGAPL - http://www.acm.org/sigapl
- АПЛ-архивы университета Ватерлоо в Канаде - ftp://watserv1.uwaterloo.ca/languages/j/Welcome.html
- Страничка фирмы APL2000 - http://members.aol.com/APL2000/index.html
- Страничка фирмы Dyadic - http://www.dyadic.com
- Новости об APL2 - http://www.software.ibm.com/ap/apl/apl2.html

- Страничка по J - http://www.jsoftware.com
- Новости в Usenet об APL и J - comp.lang.apl
- ....
- И многое другое для различных служб Internet и BBS.

Остановимся еще на **Новостях** в Internet.

Для поиска по Новостям средствами SE AltaVista нужно в окошке Search (Рис.3) установить **Usenet**, а в поле запроса снова «APL Language». В результате мы увидим хронологический протокол обмена сообщениями участников постоянной заочной конференции по языку APL. Темы там обсуждаются самые разные, например: «АПЛ-секретное оружие IBM», «АПЛ против Java», «О компиляторе для АПЛ» и т.д. Каждый пользователь Internet может принять в ней участие, послав свою ремарку или сделав «вбрасывание» по новой теме.

Несколько слов о других SE.

Система **Hot Bot** на запрос «APL Language» по Web нашла 10191 документ и вывела первые 10 названий. Здесь для каждого документа указан % соответствия введенному запросу и список начинается с самых актуальных материалов. Для каждого документа выдается краткое содержание и адрес местоположения полного текста. В первой десятке мы нашли Web-страничку журнала Vector: http://www.vector.org.uk .

В системе **Excite** мы сначала поработали в режиме DIR, выйдя на раздел «Computing», а потом запросили поиск по «APL Languge» и получили 12640 документов, сортированных по % соответствия запросу. Здесь, как показалось, меньше материалов для программиста, но есть ссылки на мощные АПЛ-приложения, например для фондовых бирж. Об этом можно судить по кратким аннотациям к документам.

Как ни странно, но через такие мощные SE как INFOSEEK и LYCOS найти информацию об АПЛ не удалось. Но нас заинтриговали результаты, полученные через слабенькую **«Русскую машину поиска»** http://search.interrussia.com , которая на запрос «Apl» выдала 26 наименований и ссылок на русские серверы, среди которых заметна активность сервера ВЦАН www.ccas.ru по вопросам АПЛ, ссылки на АПЛ в связи с фондовыми биржами, а также ссылка на список литературы по информатике: http://www.informika.ru/eng/accu/bookcase/books18.html. Эта SE вместо аннотации выдает образцы текста, в которых обнаружено ключевое слово.

Итак, надеемся, что нашу первую прогулку по Intrnet можно назвать продуктивной и мы теперь имеем опорные точки для эффективного использования информационных ресурсов и возможностей коммуникации.

Андрей Бузин

О русификации АПЛ

Вычислительный центр Российской Академии наук
Для контактов: 113525, Москва, ул.Днепропетровская, 25-30
тел. 313-49-31, E-mail buzin@ext1.ccas.ru

"Возникнет общий язык, единое средство общения всех людей"
Основы марксистско-ленинской философии, стр. 278, Политиздат, 1980

Из истории русского ⎕AV

По-видимому, предсказанию, вынесенному в эпиграф, не скоро еще предстоит сбыться. Вот и приходится мучиться с русским языком. Требование возможности использовать русские буквы одновременно с АПЛ-символами обусловлено, в первую очередь, необходимостью создания приложений для русскоязычного пользователя, во-вторую, - целями обучения языку АПЛ, в третью очередь - удобством при комментировании программ.

Любой интерпретатор АПЛ, как известно, может использовать одновременно только 256 символов, расположенных в специальной переменной ⎕AV. Таким образом, одна из проблем заключается в том, чтобы ⎕AV одновременно содержал как АПЛ-символы, так и русские (и латинские) буквы. При этом следует учесть, что позиции латинских букв, цифр и АПЛ-символов в ⎕AV практически зафиксированы, поскольку АПЛ-интерпретатор использует не начертание символа, а его позицию в ⎕AV.

Возможно, с введением нового стандарта для таблиц символов, шрифтов и ⎕AV, позволяющего одновременно использовать 32768 символов, проблема будет сведена только к составлению таблицы соответствия клавиатуры и новой большой таблицы символов[17]. Однако, пока приходится ограничивать себя 256-ю символами и договариваться об их расположении в ⎕AV.

Понятно, что перенос русскоязычных приложений с одной машины на другую, требует использования одинакового ⎕AV. Поэтому, было бы неплохо договориться о русском стандарте ⎕AV. Как оказалось, это совсем непросто. Собственно, оказалось это уже давно - в восьмидесятых годах, а сейчас подтверждается. Русификацией современного АПЛ, насколько мне известно, независимо занимались в последние 2-3 года А.Мирошников, А.Пахомов и автор этой заметки. Все они русифицировали DyalogАПЛ. Другие версии современного АПЛ либо нерусифицированы (APL2), либо русифицированы производителем (APLIII). После того, как оказалось, что ⎕AV А.Мирошникова отличается от моего, я послал Алексею предложение договориться о стандарте и опубликовать его. Возражения заключались в том, что слишком много кода уже написано с использованием его ⎕AV. Так что пока мы используем разные ⎕AV и не можем без дополнительных ухищрений обмениваться кодами. Об опыте А.Пахомова мне ничего не известно, но судя по всему, здесь ситуация такая же.

Существуют два принципиально разных подхода к расположению русских букв в ⎕AV. Первый заключается в том, чтобы добиться расположения в ⎕AV всех русских букв в порядке русского алфавита (необязательно вплотную друг к другу). Второй заключается в том, чтобы включить в ⎕AV только те русские буквы, которые по написанию не совпадают с латинскими буквами, а расположить их на тех местах, которые предназначены для символов национальных алфавитов и подчеркнутых латинских букв. Оба подхода имеют свои преимущества и недостатки.

[17] Есть еще один вариант отмирания этой проблемы: все программисты, понимающие преимущества АПЛ перед другими языками программирования, перейдут на J.

Первый хорош тем, что упорядочивание русского текста по алфавиту можно производить с помощью унарной функции ⍋ или ⍒. Поэтому, наверное, этот способ так симпатичен А.Мирошникову и А.Пахомову, часто занимающимся сортировкой. Однако у этого способа есть несколько недостатков. Во-первых, он требует дублирования в ⎕AV одинаковых по написанию символов. Это плохо по двум причинам - а)остается меньше места для других символов и б)очень трудно распознавать ошибки, связанные с использованием одинаковых символов, имеющих разное значение (символы-омонимы). Во-вторых, некоторые русские буквы приходится ставить на такие места в ⎕AV, которые объявлены производителем АПЛ резервными (в частности, не могут быть использованы в именах объектов).

Второй подход (который был реализован А.Кондрашевым и Н.Пунтиковым в знаменитой русскоязычной "семерке") избавлен от недостатков первого подхода, но не обладает его преимуществом. Однако, отказ от этого преимущества не кажется мне слишком обременительным. Дело в том, что в DyalogAPL существует бинарная функция сортировки, которая в качестве левого аргумента имеет алфавит, по которому сортировка осуществляется. Совсем нетрудно указать лишний аргумент при упорядочивании данных. Вопрос о сравнении скорости унарной и бинарной сортировок остается открытым, но предварительные эксперименты показывают, что разница в скорости, хотя и составляет 2.5 раза, не зависит от размера сортируемого массива и длины алфавита. Конечно, для разноязычных текстов придется предпринимать дополнительные усилия.

(Пожалуй, можно предложить компромиссный вариант, который, возможно, устроит сторонников как первого, так и второго подходов. Он заключается в том, чтобы расположить в ⎕AV все русские буквы в алфавитном порядке, но придать им существенно отличное от латинских начертание (например, сделать латинские прямыми, а русские - наклонными). При этом следует запретить использование нескольких русских букв (надо выбрать наименее употребительные) в именах объектов).

Пока я предлагаю пойти по второму пути и расположить русские буквы специфического начертания (46 штук) в тех позициях ⎕AV, которые могут быть использованы (имеется ввиду DyalogAPL) для составления имен объектов (обычно эти места заняты подчеркнутыми латинскими буквами и символами национального алфавита). В стандартной таблице DyalogAPL таких мест 60 штук. В качестве стандарта русского ⎕AV предлагается приведенная ниже таблица.

Места, которые остались свободными после заполнения исходных 60-ти позиций 46-ю русскими буквами отмечены символом Θ. Позиции, объявленные производителем в качестве резервных, отмечены символом ®. Позиции, в которых находится символ ©, заняты в оригинальной таблице (то есть таблице, приведенной в документации на DyalogAPL) символами, которые не употребляются в англоязычной версии (англоязычная раскладка клавиатуры не связывает с этими символами никаких клавиш).

Не существует никаких априорных предложений по поводу того, какие символы должны ставить русские любители АПЛ на места, отмеченные символами Θ, ® и ©. Следует, однако заметить, что места с символами ® лучше вообще не трогать, а два других типа мест существенно отличаются друг от друга: символы одного типа могут быть использованы в именах, а символы другого - не могут. На первые можно было бы поставить, например, греческие (или белорусские) буквы, а на вторые - дополнительные символы псевдографики.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	®	[illegible]	[illegible]	[illegible]	[illegible]	↑	%	'	α	ω
1	_	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
2	p	q	r	s	t	u	v	w	x	y	z	®	®	¯	.	θ
3	0	1	2	3	4	5	6	7	8	9	®	©	©	$	©	©
4	∆	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	®	®	©	©	©
6	⍙	Б	Г	Д	Ж	З	И	Й	Л	П	У	Ф	Ц	Ч	Ш	Щ
7	Ъ	Ы	Ь	Э	Ю	Я	б	в	г	д	ж	{	⊢	}	⊣	⎕
8	¨	з	и	й	к	⍨	л	м	н	ф	ц	ч	ш	щ	ъ	ы
9	ь	э	ю	я	θ	θ	θ	θ	θ	θ	θ	[	/	⌿	\	⍀
A	<	≤	=	≥	>	≠	∨	∧	−	+	÷	×	?	∊	⍴	~
B	↑	↓	⍳	○	*	⌈	⌊	∇	∘	(	⊂	⊃	∩	∪	⊥	⊤
C	∣	;	,	⍱	⍲	⍒	⍋	⍉	⌽	⊖	⍟	⌹	!	⍕	⍎	⍫
D	⍪	≡	≢	θ	θ	θ	θ	"	#	®	&	©	[illegible]	-	-	-
E	┼	─	┼	┤	┴	┬	┆	@	θ	θ	Ё	©	ё	·	©	©
F	:	⍷	©	©	⋄	←	→	⍝	)	]	®	®	§	⎕	⍞	©

Устройство русификатора для DyalogAPL

Сказанное выше относится к любой версии АПЛ. Далее мы будем говорить только о версии DyalogAPL, поскольку механизмы заполнения ⎕AV и определения раскладок клавиатуры различны для разных версий.

Когда символы выдаются на экран (или другое устройство вывода), их начертание заимствуется из какого-нибудь шрифта. Большинство служебных окон АПЛ-сеанса (окно протокола, окна редактирования и т.д.) устроены таким образом, что их позиции могут содержать только символы из текущего значения ⎕AV, поскольку эти позиции фактически представляют ссылки на соответствующие позиции ⎕AV. Сами же 256 позиций ⎕AV представляют собой числа-ссылки на позиции текущего шрифта. Текущий шрифт можно изменить в любой момент сеанса (см. об этом ниже), а вот соответствие между позициями ⎕AV и позициями шрифта устанавливается для сеанса один раз - при входе в этот сеанс. Оно определяется специальным файлом, указываемым в разделе "aplt" файла APL.INI, имеющем расширение .DOT и называемым "таблицей вывода"[18]. По умолчанию файлы таблиц вывода хранятся в поддиректории APLTRANS директории, в которой находится интерпретатор.

Итак, чтобы одновременно видеть и использовать русские буквы и АПЛ-символы, они одновременно должны присутствовать в текущем ⎕AV, а следовательно, и в используемом шрифте. Это означает, что мы должны иметь шрифт, который содержит все нужные нам символы: русские буквы, АПЛ-символы, латинские буквы, цифры и, возможно, служебные символы (например, двойные кавычки).

Создание шрифтов

Следует оговориться, что здесь будет идти речь только о шрифтах для Windows3.x. Дело в том, что Windows95 использует, по-видимому, несколько другие таблицы шрифтов, и, возможно, шрифты, созданные с помощью соответствующего программного обеспечения под Windows3.x не будут корректно работать под Windows95.

---

[18] Следует заметить, что таблицы вывода определяют и некоторые другие параметры сеанса, например, фоновые цвета различных подчиненных сеансу окон.

В Windows3.x шрифт является совокупностью начертаний (и правил масштабирования) 256-ти символов. Существует довольно много различных шрифтов, содержащих АПЛ-символы, как немасштабируемых (.FON), так и масштабируемых[19] (.TTF). Естественно, однако, что западные производители не позаботятся о создании шрифта, одновременно содержащего как АПЛ-символы, так и русские буквы.

Для создания русского масштабируемого АПЛ-шрифта я использовал програмное обеспечение Fontographer3.5. В качестве исходного был выбран шрифт Dyalog Alt TT, поставляемый фирмой Dyadic. С помощью Fontographer3.5 в этот шрифт на последние 64 места, были вставлены русские буквы, заимствованные из шрифта Courier New Cyr, а также на буквы "Ё" и "ё" на позиции 169 и 185 соответсвенно.

Следует заметить, что расположение русских букв в шрифте фактически предопределено. Дело в том, что перенос символов из АПЛ в другие приложения (например, с помощью буфера Windows), а также использование русских символов там, где они по умолчанию заимствуются из других шрифтов (например, из шрифта System при написании заголовков Windows-окон, создаваемых из АПЛ) требует, чтобы русские буквы располагались на тех же местах русского АПЛ-шрифта, на которых они располагаются в шрифтах, используемых другими приложениями. Большинство же русских шрифтов Windows3.x располагают русские буквы в последних 64-х позициях.

Поскольку некоторые русские буквы заняли места АПЛ-символов, последним пришлось поискать другие места в шрифте. В результате русский АПЛ-шрифт, который я назвал Dyalog Rus TT, выглядит следующим образом:

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	∧	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	⎕
⊢	⊣	⊤	⊥	←	→	↑	↓	≤	≥	⍲	⍱	⎕	⍞	⌸	⍙	⍕	∆	∇	⍒	⍋	▾	▴	⌈	⌊	⍀	≠	⊂	⊃	∩	∪	∨
	¡	×	÷	¨	⍨	⌷	§	Ё	⍝	⋄	θ	≠	≡	⍪	¯	∘	○	⌽	⍉	⊖	⍟	⍎	⎕	ё	∊	⍷	≢	⍳	⍴	⍵	⍺
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Здесь приведен вид шрифта Dyalog Rus TT, если просматривать его с помощью таблицы символов Windows3.x. Он практически совпадает с видом этого шрифта при просмотре с помощью[20] Fontographer3.5. Следует, однако заметить, что во втором случае среди первых 32-х символов можно увидеть символы псевдографики и ASCII-кавычку.

Использование этого шрифта совместно с таблицей вывода WINRUS.DOT позволяет расположить в ⎕AV символы таким образом, как это было предложено выше[21].

Шрифт Dyalog Rus TT является обыкновенным True Type шрифтом (он должен быть установлен обычным способом в Windows) и может быть использован в любых Windows-приложениях (в частности, я использую его при переносе текста из АПЛ в Word6.0).

---

[19] Заметим, что здесь речь будет идти о создании масштабируемого шрифта, поскольку использование немасштабируемого шрифта порождает дополнительные трудности при перенесении его в Word и при печати (см. мою статью в "АПЛ¨Клуб", том 1, №3).

[20] К сожалению, более поздний Fontographer4.1 дает несколько другой вид этого шрифта. С этим связаны различные трудности использования шрифта под Windows95.

[21] В действительности, в используемом мной ⎕AV нет псевдографических символов, а на местах, отмеченных символами Θ, ® и © стоят разные символы, к которым не привязаны никакие клавиши.

Шрифты, используемые в сеансе

Для того, чтобы АПЛ-сеанс использовал символы именно из шрифта Dyalog Rus TT следует указать этот шрифт в качестве свойства 'Font' для системного объекта ⎕SE. Это можно сделать вручную из сеанса с помощью выражения

```
'⎕SE'⎕ws 'Font' 'Dyalog Rus TT'
```

Вслед за названием шрифта можно указать его аттрибуты, например, размер. Если вы затем сохраните конфигурацию сеанса (с помощью команды меню Session-Save), то при следующем входе в АПЛ будет использоваться именно этот шрифт.

В принципе, все зависимые от окна сеанса объекты (потомки окна сеанса) должны иметь этот же шрифт. Если, однако, вдруг окажется, что какие-то объекты имеют другой шрифт, вы можете явно установить его для этих объектов.

Шрифт, используемый для печати на принтере объектов из АПЛ устанавливается в тексте функции ⎕SE.WSDoc.GetPref. Вы также можете изменить в функции ⎕SE.WSDoc.TimeStamp аттрибут LongDate на Date, если вас будет раздражать неправильная дата при печати объектов из АПЛ.

Раскладка клавиатуры

Реакция АПЛ-сеанса на нажатие клавиш клавиатуры определяется так называемой "таблицей ввода". Таблицы ввода представляют собой ASCII-файлы, имеющие расширение .DIN и хранящиеся обычно в поддиректории APLKEYS директории, в которой находится интерпретатор. Таблица ввода связывает клавиши клавиатуры и их комбинации с позициями ⎕AV, символы из которых появляются на экране при нажатии этих клавиш. Для того, чтобы с помощью клавиатуры можно было в АПЛ вводить русский текст, следует использовать соответствующую таблицу ввода (при наличии русских букв в ⎕AV).

Мы используем таблицу ввода RUSSIA.DIN, которая позволяет переходить на стандартную (ЙЦУКЕН) русскую клавиатуру при нажатии серого плюса. Обратный переход на латинский алфавит осуществляется так же. Статусная строка АПЛ-сеанса отображает сотояние клавиатуры.

Резюме

Итак, для того, чтобы использовать русские буквы в DyalogAPL (даже в именах объектов) и иметь возможность переносить их между АПЛ и другими Windows-приложениями, нужно воспользоваться тремя файлами (которые можно получить у автора): RUSTT1.TTF, WINRUS.DOT и RUSSIA.DIN.

Первый из этих файлов содержит шрифт Dyalog Rus TT; его удобно поместить вместе с другими шрифтами Windows и следует установить обычным способом (например, с помощью Панели управления Windows).

Второй файл следует поместить в поддерикторию APLTRANS, а третий - в поддиректорию APLKEYS директории, из которой запускается АПЛ-интерпретатор.

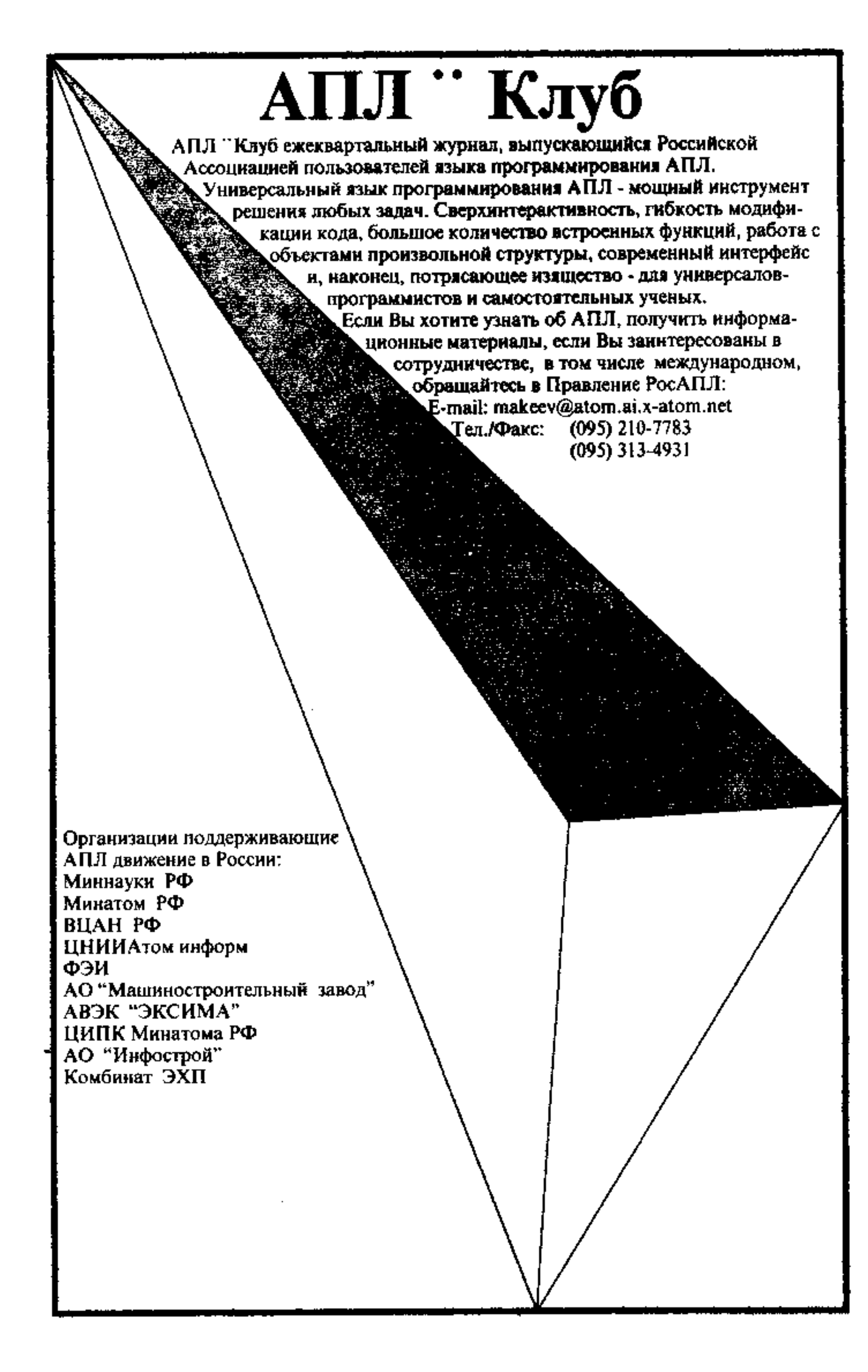
В файле APL.INI в разделе "aplt" должна быть ссылка на WINRUS.DOT, а вразделе "aplk" - ссылка на RUSSIA.DIN (этого можно добиться либо из сеанса с помощью пункта меню Options-Configuration или, проще, - вручную).

Используемый файл конфигурации сеанса (указываемый в разделе "session_file" файла APL.INI) должен содержать[22] в качестве свойства 'Font' сеанса значение 'Dyalog Rus TT' (см. выше). Лично я использую собственную конфигурацию BUZIN.DSE, которая помимо этого свойства хранит еще и картинку АПЛ-клавиатуры, а также функцию просмотра структуры данных - DISPLAY.

В результате, в DyalogAPL v.7 можно чувствовать себя вполне русским, не хуже чем в ДОС-овской версии Manugistic/Obninsk.

---

22 Файлы конфигурации не являются ASCII-файлами и указанные в них свойства сеанса лучше изменять из самого сеанса.


АПЛ ¨ Клуб
АПЛ ¨Клуб ежеквартальный журнал, выпускающийся Российской Ассоциацией пользователей языка программирования АПЛ.
Универсальный язык программирования АПЛ - мощный инструмент решения любых задач. Сверхинтерактивность, гибкость модификации кода, большое количество встроенных функций, работа с объектами произвольной структуры, современный интерфейс и, наконец, потрясающее изящество - для универсалов-программистов и самостоятельных ученых.
Если Вы хотите узнать об АПЛ, получить информационные материалы, если Вы заинтересованы в сотрудничестве, в том числе международном, обращайтесь в Правление РосАПЛ:
E-mail: makeev@atom.ai.x-atom.net
Тел./Факс: (095) 210-7783
(095) 313-4931
Организации поддерживающие АПЛ движение в России:
Миннауки РФ
Минатом РФ
ВЦАН РФ
ЦНИИАтом информ
ФЭИ
АО "Машиностроительный завод"
АВЭК "ЭКСИМА"
ЦИПК Минатома РФ
АО "Инфострой"
Комбинат ЭХП